Collaborative Resolution of Requirements Mismatches when adopting Open Source Components

Nguyen Duc Anh[1], Daniela S. Cruzes[1], Reidar Conradi[1], Martin Höst[2], Xavier Franch[3], and Claudia Ayala[3]

[1] Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim, Norway
[2] Lund University, Department of Computer Science, Lund, Sweden
[3] Technical University of Catalunya, Department of Service Engineering and Information Systems, Barcelona, Spain

Abstract. [**Context and motivation**] There is considerable flexibility in requirements specifications (both functional and non-functional), as well as in the features of available OSS components. This allows a collaborative matching and negotiation process between stakeholders such as: customers, software contractors and OSS communities, regarding desired requirements versus available and thus reusable OSS components. **[Problem]** However, inconclusive research exists on such cooperative processes. Not much empirical data exists supporting the conduction of such research based on observation of industrial OSS adoption projects. This paper investigates how functional and non-functional requirement mismatches are handled in practice. **[Results]** We found two common approaches to handle functional mismatches. The main resolution approach is to get the components changed by the development team, OSS community or commercial vendor. The other resolution approach is to influence requirements, often by postponing requirements. Overall, non-functional requirements are satisfactorily achieved by using OSS components. Last but not least, we found that the customer involvement could enhance functional mismatch resolution while OSS community involvement could improve non-functional mismatch resolution. **[Contribution]** Our data suggests that the selecting components should be done iteratively with close collaboration with stakeholders. Improvement in requirement mismatch resolution to requirements could be achieved by careful consideration of mismatches size, requirements flexibility and components quality.

Keywords: Requirements elicitation; Requirement mismatches; Open source software; Collaboration; Empirical study.

1 Introduction

The rapid growth in scale and complexity of software systems, together with the availability of third party software components, such as Commercial Off-The-Shelf (COTS) or Open Source Software (OSS) components, increase the adoption of component-based software development (CBSD) in software industry [1]. This

Anh, N.D. [et al.]. Collaborative resolution of requirements mismatches when adopting open source components. A: International Working Conference on Requirements Engineering: Foundation for Software Quality. "Requirements engineering: foundation for software quality: 18th International working conference, REFSQ 2012: Essen, Germany, March 19-22, 2012: proceedings". Berlín: Springer, 2012, p. 77-93.
The final authenticated version is available online at https://doi.org/10.1007/978-3-642-28714-5_7

adoption demands specialized software development processes that aim at supporting Off-The-Shelf (OTS, including both COTS and OSS) component acquisition, especially Requirements Engineering (RE) processes.

Traditional RE basically consists of eliciting stakeholder's needs, refining the acquired goals into non-conflicting requirements statements, and finally validating these requirements with stakeholders [2].The RE process for OSS based development is quite different from this traditional one since integration with third party components is the essential part of software development. It is an intertwined process between requirements engineering activities and OTS component selection to select the best-matched set of components and requirements. Therefore, requirements elicitation and negotiation becomes more likely a collaborative activity, which involves customers, software suppliers and third party vendors/communities. This collaborative process closely relates to the OSS component identification and selection processes [3]. The main challenge comes from the dynamic nature of requirements and evolution of OSS components [8, 4]. The continuously evolved requirements and updated versions of chosen components could make the component features differ from the requirements in post-selection phases. These mismatches between components and requirements are unavoidable and need to be resolved during the project lifetime.

Since the process of matching requirements and selected components is crucial for a successful adoption of OSS components in software projects, it is necessary to explore the relevant industrial collaboration practices, such as requirement elicitation, component selection and mismatch handling [8, 4]. Several studies have focused on the COTS component selection processes [4, 7]. However, less effort has been allocated to the investigation RE practices in the context of OSS component adoption and even less to empirical studies in this topic.

In this paper, we present a mixed quantitative and qualitative survey of how such requirement/OSS component selection and requirements mismatches are handled in fifteen European software-intensive companies in Norway, Sweden and Spain. The main purpose of the study is to explore the requirements and component selection practices and their relationships to the requirement-component mismatch resolution.

The remainder of the paper is organized as follows: Section 2 presents previous RE studies on OTS-based development. Section 3 describes our research approach. The results are provided in Section 4 and discussed in Section 5. The threats to validity and conclusions are given in Section 6 and Section 7.

2 Research background

2.1 Requirements-Components matching processes

Requirement - component matching and mismatch resolving process are overlapping activities but occurs in different phases of CBSD. While component matching consists of eliciting requirements and finding matching components in early development phase [7, 8], mismatch resolution concerns about detecting the problems with selected components and resolving it in later development [9].

Literature reveals a significant amount of research on matching process [7, 10, 11, 12, 13]. Mohamed et al. summarized the evolution of COTS selection practices in 18 COTS selection approaches [7]. The common steps include defining the evaluation criteria using requirements, COTS search, filter search results, evaluation of COTS components, and selection of best-fit COTS. Stol et al. summarized 20 different initiatives for OSS component selection and evaluation [10]. Morisio et al. surveyed 15 COTS adoption projects and characterize the COTS adoption process [11]. The common steps for the requirement phase are requirement analysis, system requirements review, COTS identification and selection, glueware and integration requirement identification. The authors also found two major issues, namely dependence on the vendor and flexibility in requirements. Paech and Reuschenbach [13] present a requirements engineering process for OSS selection. In this process, the choice of product is based on a comparison of prioritized requirements from the stakeholders and evaluation results for candidate products. Höst et al. summarize experience from a set of organizations on how to select open source components in software projects, and observe for example that it is important to understand the requirements for the identified components [12].

These studies, nevertheless do not consider the dynamic nature of requirements as well as OSS components, which lead to the issues of requirement mismatches after selecting the best-fit component at the mentioned time.

2.2 Requirements-Components mismatches resolution process

Since component features are predetermined when selecting components, the changes in requirements introduce challenges to adoption of the components. A requirement-component mismatch is a difference in functional feature or non-functional quality attributes from a given component and a desired requirement.

On one hand, some studies see requirement negotiation as an approach to resolve the mismatches [2, 8, 8, 9, 14, 15]. In these cases, the component is fixed beforehand and requirements are the target of changes [8]. Maiden and Ncube observed that this process is iterative: from an initial stage with all the customer wish-list and the full market-place available, mismatches progressively force requirements negotiation and candidate filtering until the final COTS component is selected [14]. Rolland proposed a goal-oriented approach for considering mismatches at the business level and then defined goal matching as the conceptual framework for resolving them [8]. Other approaches focused on lower level but highly challenging requirement problems, with integration requirements in call-for-tender processes [15].

On the other hand, a mismatch can be solved by modifying or adapting the selected components to fit to the requirements [4, 9, 16]. The components are modified when it takes a long time for external support [4] or when there is a need to adapt to new changes in requirement [9].

There is although a lack of empirical investigations of industrial practices on mismatch resolution. Consequently, there is no attempt to explore which approach is conducted in which scenario.

3 Research approach

3.1 Research questions

It is important to understand industrial practices on both requirement and component perspectives in order to investigate the mismatches between them in the later phases. The source of requirements and how they are described could infer how flexible the requirements can be. Besides, the component search and selection process could indicate potential problems with components while implementing requirements. The understanding of both perspectives leads to a comprehension of factors that influence requirement-component mismatches. This argument leads us to RQ1:

RQ1: What are the general practices of requirement elicitations and OSS component selection in OSS adoption software projects?

Secondly, we distinguish the concepts of functional and non-functional requirements with regarding to requirement mismatches. In this study, we define functional mismatches as the differences between functional requirements and features provided by the components. These functional mismatches are investigated in the component level. Since the functional requirements are often explicitly described, it is not problematic to identify the functional mismatches when they occur. We are interested in investigating how the functional mismatch between a requirement and a component is handled by project stakeholders. It is hypothesized as an intertwined process of negotiation and technical resolution that involve customer, developers and OSS community. To investigate this scenario in industry, we propose the RQ2:

RQ2: How are the functional mismatches between requirements and OSS components collaboratively managed in OSS adoption software projects?

Thirdly, in addition to discovering what functionalities are important to users at the system level, qualities associated with particular functionality/user goals should be elicited. The qualities may need to be translated by developers from user-level objectives, values and concerns into specific technical quality requirements, though non-functional requirements are often not well-described and poorly understood [17, 18], hence the mismatches between non-functional requirements and components are hard to investigate and assess. Besides, non-functional requirements are normally system characteristics. Therefore, they are often verified in the later phases of system development, when the modules are integrated and tested. Consequently, instead of investigating the mismatches between non-functional requirements and components, we investigated which and how non-functional requirements are fulfilled by using OSS components. This rationale leads to RQ3:

RQ3: How are non-functional requirements fulfilled by using OSS components in OSS adoption software projects?

3.2 Data collection and analysis

The study was performed in the period between September 2010 and September 2011, including study design, piloting, data collection and analysis.

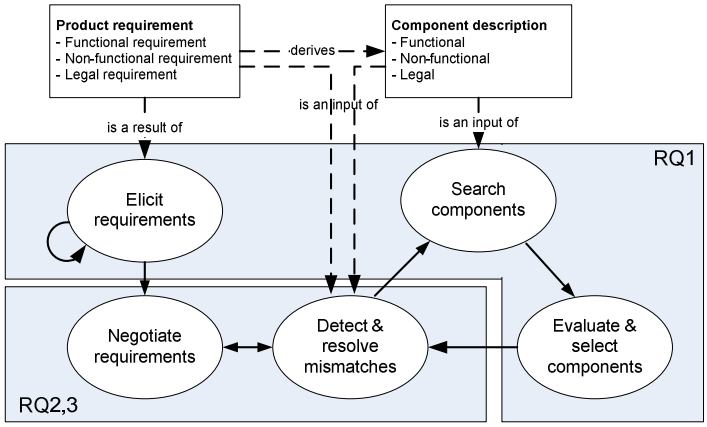


Figure 1: Research questions mapping

Population: Our target is software-intensive organizations that adopt OSS in producing software product. This population includes organizations with different sizes and in different application domains. 64 companies from our contact list were selected and contacted by phone call and email, in which fifteen stakeholders (developers or project leaders), who represented for 15 projects, agreed to participate in the survey. Some of the contacts were not eligible for participating due to several reasons, such as lack of adoption of OSS components in the projects, the companies changed the OSS adoption policy or the adoption strategy was not publishable.

Interview guide (survey): The method used in this study is semi-structured interviews. The interview guide was adjusted after three pilot interviews. The purpose of the survey is to discover the practices in OSS adoption, such as Requirements elicitation, Component selection, Requirement mismatch resolution and Collaboration process in adopting OSS components. In the scope of this study we focused on results extracted on RE practices. The survey was designed as a 5-section survey, with both closed and open questions. The closed questions were used to solicit information on interviewee and project context. The open questions were used to gather information on component-requirement mismatches resolution practices and communication to the community. The survey also included explanation for important terminology and description of context background in order to offer a common understanding for all participants. The relevant survey questions are given in the Appendix.

Data collection procedure: The interview survey was sent to all participants some days before the interview meeting. In this way, the participants could be well-prepared for the interview. The participants were asked to fill in the first two parts of the survey and give back to us before hand. The next three parts of the survey were asked directly to the participant during the interview. Each interview session lasted between 40 to 75 minutes. Interviews were attended by one to three interviewers. The conversations were recorded and transcribed for posterior analysis. The transcripts vary from 13 to 21 pages in size.

Analysis procedure: We analyzed the filled-in questions and transcripts using a qualitative research tool NVIVO. The approach is a tailored thematic synthesis [19]. The analysis consists of four steps: extracting data from the interview transcription; grouping data into fundamental groups based on the structure of the survey; coding data within each category; translating codes into themes and linking relevant themes together. The first two authors examined the categories from different perspectives and searched for explicitly stated or concealed opinions about how Requirement-Component mismatches are handled in industry. The results from the analysis are described in Section 4. For each research question, we conducted a quantitative summary of answers on closed questions from each interview and qualitative analysis of taped conversations to support the quantitative part.

4 Results

4.1 Projects description

We surveyed the requirement mismatch resolution process in fifteen projects from Norway, Sweden and Spain. Table 1 shows some of the projects characteristics of the surveyed projects. The team size ranges from two to 250 people. The project life cycles include ad hoc development, waterfall, iterative development and agile, with a prevalence of the agile model in seven projects. The adoption of lightweight development life cycles, such as Agile or Scrum, introduces flexibility in requirements elicitation and component selection. The application domain covers a wide variety of domains, including Communication system, Information system, Web application and Public-sector support, with a dominant of Public sector support in five cases.

Table 1: Projects characteristics

ID	Team size	Development process	Application domain	OSS portion	Selection in RE?	Req. source
P1	20-25	Iterative	Communication system	90%	Yes	External
P2	4	UNK	Audio/ Video processing	10%	Yes	Internal
P3	2	Agile	Search engine	80%	Yes	Internal
P4	18	Waterfall + Scrum	Embedded system	ca. 17000 KLOC	Yes	Internal
P5	2	Iterative	Oil/gas support product	77%	No	Internal
P6	200	Scrum	Public sector support	75%	No	External
P7	4	Scrum	Document processing	10%	Yes	External
P8	20	Agile	Public sector support	66%	Yes	External
P9	2	Agile	Information system	90%	No	External
P10	2	Iterative	Public sector support	60%	Yes	External
P11	250	Agile	Telecommunication	90%	No	External
P12	3	Ad-hoc, requirement-driven	University	90%	No	External
P13	3	Ad-hoc	Information system	5%	No	Internal
P14	5	Tailored waterfall	Public sector support	80%	Yes	External
P15	6	Iterative	Public sector support	20%	No	External

The OSS components portion represents the interviewees' estimation about the proposition of actual use part of OSS components in total product size in LOC. The OSS portion ranges from 10 to 90%. In one project, the interviewee could not provide a percentage due to absent information about the total product size. The large portion of OSS shows the importance of OSS components in the software, which could influence the priority of components during the mismatch resolution process.

The last column indicates whether the component selection is decided in the RE phase or not. Interestingly, in seven projects, the components selection is not considered in the RE phase. In projects 5, 6, 13 and 15, the requirements are predetermined (i.e. subcontract or outsourcing) and selecting components are considered and design or coding level as an approach to implement given requirements. In Project 9, the company provides services to customers and selection of components is transient in RE phase.

4.2 RQ1: What are the general practices of requirements elicitation and OSS component selection in OSS adoption software projects?

4.2.1. Requirements elicitation practices

Source of requirements: In eight projects, requirements come from external customers, and in one of the cases, managed by an external consulting company, as shown in Table 1. In one project the requirements come from both external customers and internal development team since customers required a system with similar functionalities of existing system. In this project, the requirements are flexible since the customers require the product to confront a predetermined standard and development team has to find out the detail requirements themselves.

In five projects, requirements are market-driven, coming from an internal development team. In three of them, developers also play the role of customers. Moreover, in the fourth one, they consulted other development teams that deployed similar systems, whilst in the fifth case the marketing department also had a stake. Another project's requirements come solely from the marketing department. In this project, the software is a part of an embedded system to sale.

Requirement description level: Figure 2 shows that among investigated projects, seven projects have requirements coarsely described. We categorize the requirement specification according to three categories: coarsely, medium and detail based on requirement description and notation. The detail level of requirement specification infers the flexibility of the requirements since the coarser one is probably the more flexible one. The coarse description of requirements in major projects is probably caused by the adoption of agile methodology. Only three projects have requirements described in detail and three in-between. Concerning specification notations, free text is used as much as structured text. Both of these requirement notations are used in seven projects. Use cases and test cases are used in three projects each, and one case used "informal" flow diagrams for expressing navigational-related requirements in a web application.

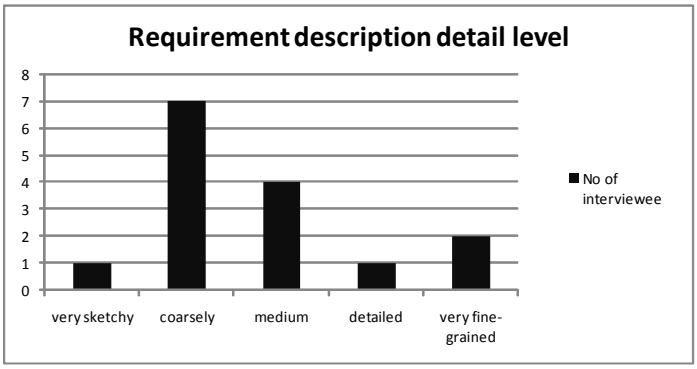


Figure 2: Requirement description detail level

4.2.2. Component identification and selection practices

Component identification: Figure 3 describes the approaches to identify the OSS components in company's projects. Projects often used more than one approach. The most common approach is based on previous experiences without formal search and evaluation processes, which are used by ten out of fifteen interviewees. The second option (8 out of 15 interviewees) is either to use a search engine or to ask friends, colleagues or someone that has experience from before with the component. Both of these options were six interviewees mentioned about peer-review or grey literature as another source to find components. Only two projects contact customers during component identification process. One of the interviewees could not provide details in this questions nor the rest of this subsection since the selection process was entirely run by a team of software architects.

Component selection process: none of the interviewees reported the usage of formal evaluation processes, which are abundant in literature [7, 10], in their projects. This observation is similar to findings from a previous study [20]. The evaluation activity is normally undertaken in ad hoc manner. For small components, reading the documents or looking into the code is probably sufficient. For the more significant components, a survey may be conducted to search for alternative options. A short trial with the goal to "try to get it work as a proof-of-concept" is also one possibility.

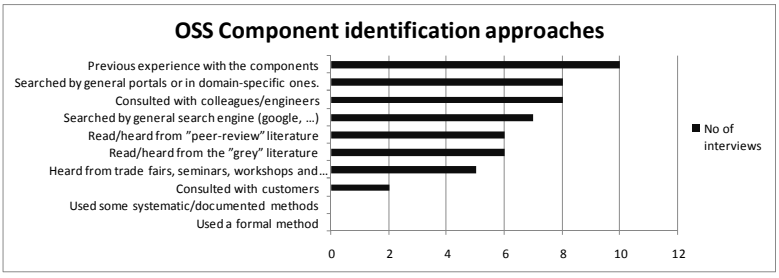


Figure 3: OSS Component - Requirement identification approaches

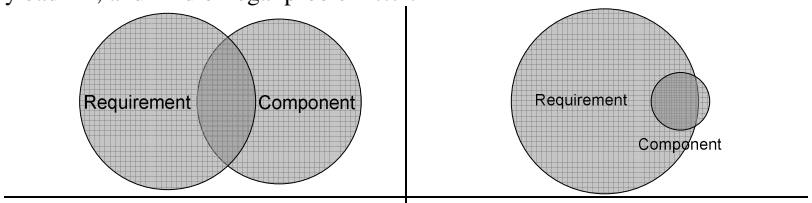
4.3 RQ2: How are the functional mismatches between requirements and OSS components collaboratively managed in OSS adoption software projects?

4.3.1. Functional mismatches identification

Grounded from interview's conversation, there are three main criteria used to decide on a mismatch between a requirement and an open source component, namely fit to functional requirements, fit to non-functional requirement and fit to legal requirement. As the basic purpose of using external components, the OSS components should have the basic functionalities that fit to the requirements. The functional mismatch is the ratio between part of the component that satisfies the requirement and the full set of requirement features. In case of small or fine-grained requirement (as in Figure 4a), the mismatch appears when there is a relative small portion of overlap functionality between the requirement and component. In case of large or coarse-grained requirement or product feature (as in Figure 4b), the mismatch happens when the component only provide part of required requirements.

With respect to non-functional requirements, reliability of the components is a highly cited criteria, and concerns the number of defects in the component; if the component is functionally fit to the requirement, but it contains many bugs then it would take a time and effort to use the components.

Last but not least, third criteria concern about component license issue. OSS components employ different types of licenses that would be taken into consideration, as one interviewee mentioned: "a lot of GPL license components cannot be used … doing a mistake like shipping a GPL license component in a commercial product is very bad PR, and kind of legal problem …".



(a) Fine-grained requirement (b) Coarse-grained requirement

Figure 4: Functional mismatch type

4.3.2. Mismatches resolution approaches

Figure **5** provides the scenarios in which mismatches are handled. The majority answered that they change the components in some way, such as creating a glueware or addware, modifying the components and replacing the components, rather than get requirements affected. Nine interviewees said to modify or add adjustments to the OSS components by themselves. Six of them chose to make the changes globally, and send it back to the OSS community. Three interviewees make the changes locally, which are reserved for internal use only. Only two interviewees utilize community support for adapting the components while three interviewees chose commercial vendors instead.

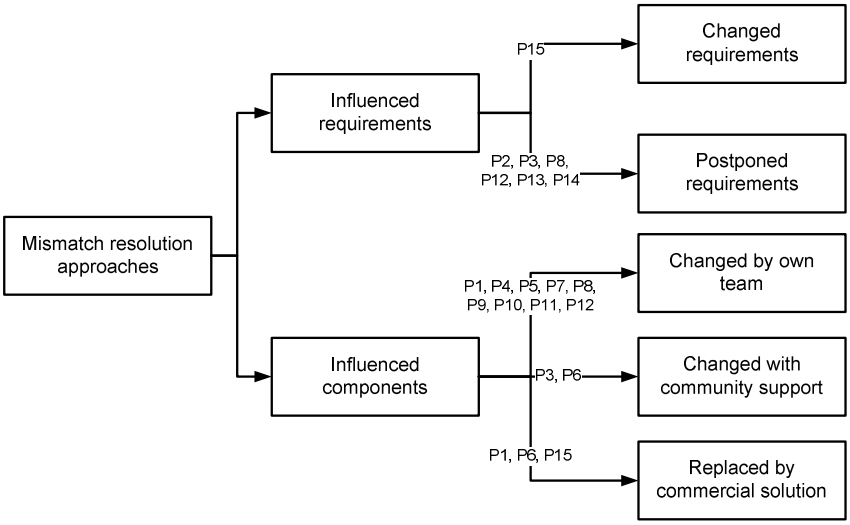


Figure 5: Requirement - component mismatch resolution approaches

4.3.3. When are requirements changed?

In most of the cases, the requirement is not a subject to change or relax as it is often at the higher priority over components. Some interviewees said: *"… there is no case giving up on the requirements. Requirements are usually at first priority", "… selecting an OSS component does not impact the requirement so much. It is not so much you can relax your requirement a bit or replace five hour of coding with existing component, it is not possible." "… normally requirement is not in the position to relax it a lot.", "... requirements were not negotiated because the project was about reengineering a legacy system into a web application; the requirements were the ones for the departing system".*

In three projects, requirements come from predefined standards, government reform and they are not possible to negotiated or modified. In some other projects, the adopted components are of small to moderate size, and are implemented by domain specific libraries or as part of a framework. Since the integrated components serve for small and fine-grained functional requirement, it does not affect much on the overall requirements of the system. Some interviewees said: *"requirements usually do not really affect choice of components that much, as most components we use are small and not visible to the customer", "… we use smaller components rather than larger sort of application server or something, the customer doesn't really see the component as a separate components, it is a part of the product".*

Besides, OSS components offer an opportunity to modify/adjust the components upon the mismatches. This flexibility of OSS components gives more chances to satisfy the requirements, as some interviewees said: *"If there is a partial mismatch, I think we just use it for what we could use it for.",* "*…was quite simple to extend the open source project to get the functionality we needed …*", *"… one of the reasons to*

select one of the components was that it provides a proprietary script language that allows specifying its behavior when starting the system". Particularly, in one project, the mismatched component was rewritten from the scratch since it was a small library.

We found only one case where the option of relaxing requirements was selectively taken. The development team adopted a compensatory strategy: whilst explaining to the customer which (non-critical) requirements were not satisfied, they emphasized additional functionalities that the OSS component was covering and could be incorporated into the delivered system. It was also helpful that the customer had a very technical profile and was able to understand the consequences (in terms of cost) of not relaxing the requirements.

4.3.4. When are requirements postponed?

While there is only one case where requirements is relaxed or modified, it is worth-noticed that seven interviewees mention scenarios where some requirements were postponed. The requirements were postponed in some critical cases. In one case, requirements were postponed due to the quality of components: *"... we have postponed the project because there are a lot of bugs in [Component name]. We have to look for a new library"*. In the other case, the customer accepted to postpone some non-essential requirements, the strategy followed by the development team to convince the client was to highlight those features that were not required by the customer and were offered by the component.

4.4 RQ3: How are non-functional requirements fulfilled by using OSS components in OSS adoption software projects?

Figure 6 shows the perceptions of interviewee about non-functional requirements achieved by using OSS components. For each of non-functional requirement attribute, the grey column represents for the number of interviewees that mentioned about it. The black column shows the number of interviewees that satisfy with the quality attribute of the OSS component. As shown in Figure 5, the most concerned non-functional requirements regard to OSS components are performance, reliability, maintainability and cost. The list of concerned non-functional requirements in our study is different from the most concerned requirements in Berntsson Svensson et al., namely usability, performance and flexibility [21]. Their context was limited to the embedded system and market-driven projects and it may be the reason for the conflicting results.

4.4.1. Performance

Performance is satisfied by using OSS components in nine out of eleven interviews. The performance is perceived as sufficient or at least not affecting much the overall performance of the system. Some interviewees mention the problem with performance problems but these mainly come from hardware and infrastructure issues.

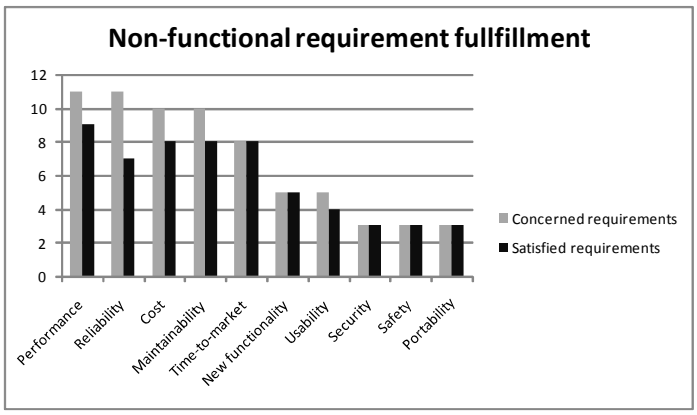


Figure 6: Non-functional requirement fulfillment by OSS components

4.4.2. Reliability

There are contradictory opinions about reliability of OSS component. Seven interviewees experienced good reliability, with little or few bugs, with the correctness of the system exceeding expectation. Four interviewees had experiences with both reliable and unreliable OSS components. There is a misunderstanding during the conversations with some interviewees between Reliability and Maintainability. Some people said the OSS component turned out to have sufficient reliability because the code is available and then it is easy to fix the bug.

4.4.3. Maintainability

Maintainability is an important feature for OSS components. Eight out of ten interviewees are satisfied with the maintainability of the components. The factors that that contribute positively to the maintainability of OSS components are:

- The openness of the code, that allows developers to "dive into the code" to fix bugs.
- Synchronization with the upstream development: the OSS community offer a chance for the company to escape from the burden of maintenance since the components can be synchronized with the upstream development, contact with the OSS community (in comparison with a commercial component), significantly influence the maintainability of the components.
- Documentation of the code that facilitate understanding and using components.

As maintainability is as important as reliability for selecting suitable OSS components, practitioners should look for components that are not only reliable but also has a high bug fixing rate.

4.4.4. Time and cost

Concerning time and cost**,** all of the interviewees are happy with the reduction of deliver time by using OSS components. Eight out of ten interviewees are happy with the cost due to the saving of licensing and implementation. There are two cases where cost is not satisfied. In one project, a lot of problems were reported due to the technical misuse of OSS component. At the end the team had governance problems that resulted in higher costs and poor reliability, performance and particularly maintainability, because the team in charge was not very big and the learning curve too steep.

5 Discussion

Our observations from fifteen projects with different context settings and requirement practices offer some implications for improvements in requirement mismatch handing process. The findings are consolidated in five propositions.

Proposition 1: market driven requirements are more flexible than bespoken requirements while resolving functional mismatches in OTS based development.

The result suggests that the choice of requirement mismatch handling approaches varies across projects and most likely do not depend on project context factors, such as: team size, application domain, development life cycle, portion of OSS components and component selection phase. Therefore, the decision whether to modify OSS components or influence requirements is influenced by the nature of requirements and components themselves, e.g. type of requirement source. Among five projects with requirements from internal development teams, four of them have requirements postponed. The requirements from internal teams (or market-driven type of requirement) would be more flexible due to consideration of given functionality and implementation effort. The requirements from external customers (or bespoken type of requirement) are less flexible due to contractual predetermination in required functionality.

Proposition 2: A functional mismatch with a flexible requirement is resolved by postponing the requirement, rarely by changing it.

Although flexibility of requirement does not hinder the requirement priority, it is beneficial for mismatch resolution by extending the resolution time. Regardless of requirement source type, requirement is normally in the first priority. Therefore, the definition of requirement flexibility is associated with the ability to postpone requirements, rather than with the ability to change or give up on the requirements [11]. Postponing requirements often occurs with customer negotiation and debugging process.

Proposition 3: A small functional mismatch is resolved by modifying OSS component while a large functional mismatch is resolved by replacing it by another OSS component or a COTS one.

Our data suggests that the detail level of requirement and the size of components influence how mismatches are resolved. Given the flexibility of OSS components, the small mismatch (a fine-grained requirement with small component) require less effort to modify or rewrite while a large mismatch take much more effort to close the gap by adapting the components. This observation recommends that component selection in

early phase, such as requirement elicitation, would be risky when the requirement is not clear enough and in general level. However, selecting components for fine-grained requirements in later phase, such as design or implementation also have threats of extra cost in integrating small components.

Proposition 4: Component reliability issues lead to postponed requirements by fixing the component or replacing it.

Reliability is one of the most concerned non-functional attributes while adopting OSS component. It also receive contradict perception from interviewees. It is difficult to correctly evaluate component reliability in component selection phases. The information that are used as early quality indicators and selection criteria, such as number of fixed bugs, component reputation and project roadmap, is not sufficient. The problem in this non-functional attributes would influence functional requirements by delaying the accomplishment of these requirements. The fewer bugs in components would take more time to fix while many bugs in components would require for the replacement. In later case, the selection and matching process will be conducted again, which cost much more time and effort. This suggests a better care of non-functional requirements of OSS components when selecting components.

Proposition 5: A functional mismatch that gets support from the OSS community is associated with a perceived increase in satisfaction regarding component maintainability.

Three collaborative resolving requirement mismatch involve customers, OSS community and commercial vendor, alternatively. Keeping changes in components synchronized with OSS community is beneficial for fixing and maintaining these components. In resolving requirement mismatch, community involvement would not only reduce the developer's effort in maintaining the components but also bringing more confidence on component quality as "given enough eyeballs, all bugs are shallow". As maintainability is as important as reliability for selecting suitable OSS components, practitioners should look for components that are not only reliable but also has a high bug fixing rate.

6 Threats to validity

In this study, most variables are taken directly, or with little modification, from the existing literatures. To ensure that the given concepts are understood correctly by the interviewees, we sent the interview guide with a detailed description of the survey to the interviewee beforehand. One of the possible threats to the internal validity is our misunderstanding of respondents' answers. Although at least two interviewers carried out the interviews and there was only one interviewee in each interview, we taped all interviews. Listening to the tape helped to ensure correct interpretation of answers and comments. However, having an independent (third) person to listen to the tape might increase data quality. During the interview, we tried to ensure the interviewee understand what they are asked. The primary threat to external validity is that the study is based on few and possibly not typical projects. In general, most empirical studies in industry suffer from non-representative participation. In the data sampling step, we

tried to have projects with all sizes, from various domain application and have different portion of OSS adoption in the projects. Besides, this study is still a preliminary study. Future studies with more interviews will be implemented to give more statistically significant results.

7 Summary and future works

The main purpose of this study is to gain understanding of how requirements mismatches are collaboratively handled in OSS adoption projects. We found two scenarios in solving functional mismatches. The main resolution approach is to get the components changed by the development team themselves, OSS community or commercial vendor. The choice of adapting or replacing components depends on the mismatch size, component reliability and level of community support. The other resolution approach is to influence requirements, often by postponing requirements. This scenario is associated with issues of component reliability and maintainability. Non-functional requirements are satisfactorily achieved by using OSS components in general. Finally, we found that the customer involvement enhance functional mismatch resolution while OSS community collaboration could improve non-functional mismatch resolution.

The study identifies topics for future research on the requirement mismatches handling process. One of the potential future extensions of the study is a supporting framework for OSS component selection decision-making. The main purpose of the framework is to find out indicators of components reliability and maintainability from the OSS component community. Besides, some of the context factors show potential impact on requirement mismatch resolution decision, such as source of requirement or reliability of components. However, we do not have enough data to conduct a quantitative analysis on these factors. In future studies with more data points, a more quantitative analysis of impacting factors could be implemented. Last but not least, we highlighted the importance of stakeholder involvement in mismatch resolving process. The deeper understanding of stakeholder involvement would help to improve the matching process.

Acknowledgements

This work has been supported by the Spanish project TIN2010-19130-C02-01 and partly funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering, (http://ease.cs.lth.se).

Reference

1. Ø. Hauge, C. P. Ayala, and R. Conradi, "Adoption of Open Source Software in Software-Intensive Industry - A Systematic Literature Review", *Information and Software Technology*, 52(11), pp.1133-1154 (2010).

2. C. Alves, "COTS Based Requirements Engineering", *Component Based Software Quality*, LNCS 2693, pp. 21 -39, 2003.
3. A. Parra, C. Seaman, V. Basili, S. Kraft, S. Condon, S. Burke, and D. Yakimovich,“ The Package-Based Development Process in the Flight Dynamics Division”, 22nd Software Engineering Workshop, NASA/Goddard Space Flight Center, pp. 21-56, 1997.
4. C. Alves, A. Finkelstein, “Negotiating Requirements for COTS-Based Systems”, 8th Int. Workshop on Requirements Engineering: Foundation for Software Quality, 2002, Essen.
5. J. Li; R. Conradi, C. Bunse, M. Torchiano, O. Slyngstad, M. Morisio, "Development with Off-the-Shelf Components: 10 Facts," *IEEE Software*, 26(2), pp.80-87, March-April 2009.
6. M. Morisio, C. B. Seaman, V. R. Basili, A. T. Parra, S. E. Kraft, and S. E. Condon, “COTS-based software development: processes and open issues,” *Journal of System and Software*, vol. 61, no. 3, pp. 189-189, 2002.
7. A. Mohamed, G. Ruhe, and A. Eberlein, “COTS Selection: Past, Present, and Future,” 14th IEEE Int. Conf. on the Engineering of Computer-Based Systems, pp. 103-114, 2007, Tucson.
8. C. Rolland, "Requirements Engineering for COTS based Systems", *Information and Software Technology*, vol. 41, pp. 985-990, 1999.
9. A. Mohamed, G. Ruhe, and A. Eberlein, “MiHOS: an approach to support handling the mismatches between system requirements and COTS products,” *Requirement Engineering*, vol. 12, no. 3, pp. 127-143, 2007. 5
10. K. Stol, M. Ali Babar, “A Comparison Framework for Open Source Software Evaluation Methods”, *IFIP AICT*, 319, pp. 389-394, 2010.
11. M. Morisio, C. B. Seaman, A. T. Parra, V. R. Basili, S. E. Kraft, and S. E. Condon, “Investigating and improving a COTS-based software development”, 22nd International Conference on Software Engineering, pp. 32-41, 2000, Limerick.
12. M. Höst, A. Orucevic-Alagic, P. Runeson, “Usage of Open Source in Commercial Software Product Developmemt – Findings from a Focus Group Meeting”, PROFES, pp. 143-155, 2011, Torre Canne.
13. B. Peach, B. Reuschenbach, “Open Source Requirements Engineering”, 14th International Requirements Engineering Conference, pp. 252-259, 2006, Minnesota.
14. N.A.M. Maiden & C. Ncube, “Acquiring Requirements for Commercial Off-The-Shelf Package Selection”, *IEEE Software*, vol. 15(2): 46-56, 1998.
15. S. Lauesen, “COTS tenders and integration requirements”, *Requirements Engineering,* vol. 11(2), pp. 111-122, 2006.
16. J. Li, R. Conradi, O. P. N. Slyngstad, C. Bunse, C., M. Torchiano, and M. Morisio, “An Empirical Study on Decision Making in Off-the-shelf Component-based Development”, Proc. 28th International Conference on Software Engineering, pp. 897-900, May 2006, Shanghai.
17. L. Chung, B.A. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Norwell, 2000.
18. S. Jacobs, “Introducing Measurable Quality Requirements: A Case Study,” 4th ISRE 99, IEEE Comput. Soc, pp. 172-179, 1999.
19. D. S. Cruzes and T. Dybå, “Recommended Steps for Thematic Synthesis in Software Engineering”, 5th Empirical Software Engineering and Measurement, 2011, Banff, Canada.
20. C. P. Ayala, Ø. Hauge, R. Conradi, X. Franch and J. Li, "Selection of Third Party Software in Off-The-Shelf-Based Software Development - An Interview Study with Industrial Practitioners", *Journal of Systems and Software*, 84(2011), pp. 620-637.

21. R. Berntsson Svensson, T. Gorschek, B. Regnell, "Quality Requirements in Practice: An Interview Study in Requirements Engineering for Embedded Systems", *Lecture Notes in Computer Science,* Springer Berlin, vol 5512, pp. 218-232, 2009.

Appendix

Part 1: Background Questions on Project and System (to be filled up prior to the meeting)

1.1 What was the mean annual staff-size of the project (both full- and part-time employees)?

1.2 What part of the staff had previous experience with OSS-based development?

1.3 Did you have previous experience with OSS-based development before joining the project?

1.4 What was the total effort of the project?

1.5 What was (roughly) the starting time of the project?

1.6 What was the time of the first complete delivery from the project?

1.7 What were the major application domain(s) of the system?

1.8 Where did the requirements come from?

1.9 How were the functional Requirements described with regard to level of detail?

1.10 What was the overall, software development process/environment of the project?

Part 2: Identify initially some OSS Component candidates that may satisfy the Requirements

2.1 In which lifecycle phases were such OSS Components selected?

2.2 How was the search process and initial evaluation for such OSS Components done?

2.3 What were the main information sources in deciding whether the OSS Component candidates from point 2.2 could (partly) match your functional Requirements?

Part 3: Final evaluation and decision process to resolve possible Requirements mismatches vs. OSS Components

3.1 What did you do when the functional Requirements could not be sufficiently matched by OSS Component candidates?

3.2 How well were the major non-functional Requirements ("quality attributes") achieved?

3.3 Focusing on the 5 most important functionalities from the Requirements, can you name and explain the matching OSS Components that you finally integrated into your system?

3.4 How big part of the system do the OSS Components now occupy?